

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 РАЗРАБОТКИ СИСТЕМЫ УПРАВЛЕНИЯ
АВТОЗАПРАВОЧНОЙ СТАНЦИЕЙ НА
PYTHON
по дисциплине
«Информатика и программирование»

Студент
гр. БИН-25-3 _____ М.В. Кирийчук
Ассистент
преподавателя _____ М.В. Водяницкий

Содержание

1 Введение	3
1.1 Принципы проектирования	3
2 Выполнение работы	4
2.1 Архитектура программы.....	4
2.2 Инициализация станции	4
2.3 Обслуживание клиентов	5
2.4 Пополнение и перекачка топлива	6
2.5 Аварийный режим	7
2.6 Сохранение данных	8
3 Тестирование	9
4 Заключение	10

1 Введение

Автоматизация процессов на автозаправочных станциях (АЗС) играет ключевую роль в современном бизнесе. Консольные системы, используемые для этих целей, позволяют эффективно контролировать запасы топлива, вести учет продаж, оперативно реагировать на технические неисправности и обеспечивать бесперебойную работу станции.

Наша цель — разработать простую и надежную систему управления АЗС, которая будет функционировать без сбоев, будет интуитивно понятной и сможет эффективно выполнять все возложенные на нее задачи.

1.1 Принципы проектирования

При разработки были заложены следующие принципы:

- Надежность: все операции проверяются на корректность, некорректный ввод не приводит к аварийному завершению.
- Состоятельность: система сохраняет полное состояние между запусками.
- Безопасность: аварийный режим блокирует все операции и требует ручного восстановления.
- Модульность: код разделён на логические компоненты, каждый из которых отвечает за свою зону ответственности.

Эти принципы обеспечивают соответствие требованиям промышленного программного обеспечения.

2 Выполнение работы

2.1 Архитектура программы

Программа реализована в виде единого класса FuelStation, который инкапсулирует всю логику управления станцией. Основные компоненты:

- Данные цистерн: словарь cisterns с параметрами (тип, объём, состояние).
- Цены на топливо: словарь fuel_prices.
- Колонки: словарь pumps с привязкой видов топлива к цистернам.
- Финансы и статистика: баланс, количество обслуженных автомобилей, объемы продаж.
- История операций: список history для аудита.
- Аварийный режим: флаг emergency_mode.

На рисунке 1 представлен код структуры класса FuelStation.

```

1 class FuelStation:
2     def __init__(self):
3         self.cisterns = {}
4         self.fuel_prices = {
5             'AI-92': 47.50,
6             'AI-95': 51.20,
7             'AI-98': 58.30,
8             'DT': 56.00
9         }
10        self.pumps = {}
11        self.balance = 0.0
12        self.stats = {
13            'cars_served': 0,
14            'fuel_sold': {'AI-92': 0, 'AI-95': 0, 'AI-98':
15            0, 'DT': 0},
16            'income_by_fuel': {'AI-92': 0, 'AI-95': 0, 'AI
-98': 0, 'DT': 0}
17        }
18        self.history = []
19        self.emergency_mode = False
20
21        self.initialize_data()

```

Рисунок 1 – Основная структура класса

Такой подход позволяет управлять всеми аспектами станции в рамках одного объекта, что упрощает сериализацию и восстановление состояния.

2.2 Инициализация станции

При первом запуске создаётся стандартная конфигурация, соответствующая техническому заданию:

- 1) 5 цистерн с разными типами топлива (AI-92, AI-95, AI-98, DT) и начальными уровнями.
- 2) 8 колонок с заданными привязками к цистернам.

На рисунке 2 представлена инициализация цистерн.

```

1 def initialize_data(self):
2     self.cisterns = {
3         'AI-92_1': {
4             'type': 'AI-92',
5             'max_volume': 20000,
6             'current_volume': 12400,
7             'enabled': True,
8             'min_level': 1000
9         },
10        'AI-95_1': {
11            'type': 'AI-95',
12            'max_volume': 20000,
13            'current_volume': 9800,
14            'enabled': True,
15            'min_level': 1000
16        },
17        # ... остальные цистерны
18    }

```

Рисунок 2 – Инициализация цистерн

Метод `initialize_data` заполняет структуры начальными значениями, после чего вызывается проверка уровня топлива для автоматического отключения цистерн с низким запасом.

2.3 Обслуживание клиентов

Метод `serve_customer` реализует полный цикл продажи:

- Проверка аварийного режима.
- Выбор доступной колонки.
- Выбор типа топлива с учётом состояния цистерны.
- Ввод количества литров и проверка достаточности запаса.
- Расчёт стоимости и подтверждение оплаты.
- Списание топлива, обновление статистики, запись в историю.

На рисунке 3 представлен код логики продаж.

```

1 def serve_customer(self):
2     if self.emergency_mode:
3         print("Ошибка: Станция находится в аварийном режиме!")
4         return
5
6     # Выбор колонки
7     for pump_num in sorted(self.pumps.keys()):
8         available_fuels = []
9         for fuel_type, cistern_id in self.pumps[pump_num]['fuels'].items():
10            if self.cisterns[cistern_id]['enabled']:
11                available_fuels.append(fuel_type)
12
13    # Проверка оплаты и списание топлива
14    confirm = input("Подтвердить оплату? (y/n): ").lower()
15    if confirm == 'y':
16        self.cisterns[cistern_id]['current_volume'] -=
17        liters
18        self.balance += price
19        self.stats['cars_served'] += 1
20        self.log_operation('SALE', f"Колонка {pump_choice}, {fuel_type}, {liters}, {price:.2f}₽")

```

Рисунок 3 – Основная логика продажи

Все этапы защищены проверками ввода, что исключает аварийное завершение при ошибках пользователя.

2.4 Пополнение и перекачка топлива

Операции пополнения `refuel_cistern` и перекачки `transfer_fuel` реализованы с учётом бизнес-ограничений:

- 1) Пополнение: проверка на превышение максимального объёма цистерны рисунок 4.
- 2) Перекачка: разрешена только между цистернами одного типа топлива; контролируются достаточность топлива в источнике и свободный объём в приёмнике. Рисунок 5

```

1 def refuel_cistern(self):
2     new_volume = cistern['current_volume'] + liters
3     if new_volume > cistern['max_volume']:
4         print(f"Превышение максимального объема! Максимум: {cistern['max_volumel']}")"
5         return
6     self.cisterns[cistern_id]['current_volume'] = new_volume
7     self.log_operation('REFUEL', f"Цистерна {cistern_id}, добавлено {litersl}")

```

Рисунок 4 – Пополнение цистерны

```

1 def transfer_fuel(self):
2     # Проверка типа топлива
3     if self.cisterns[source_id]['type'] != self.cisterns[
4         dest_id]['type']:
5         print("Ошибка: перекачка возможна только между
6             цистернами одного типа!")
7         return
8
9     # Проверка объемов
10    if liters > source_cistern['current_volume']:
11        print("Недостаточно топлива в источнике!")
12        return
13
14    if dest_cistern['current_volume'] + liters >
15        dest_cistern['max_volume']:
16        print("Превышение максимального объема в приемнике!")
17        return
18
19    # Выполнение перекачки
20    self.cisterns[source_id]['current_volume'] -= liters
21    self.cisterns[dest_id]['current_volume'] += liters
22    self.log_operation('TRANSFER', f"Из {source_id} в {
23        dest_id}, {liters}л")

```

Рисунок 5 – Перекачка топлива

Обе операции логируются в истории.

2.5 Аварийный режим

Аварийный режим имитирует критическую ситуацию (утечка, пожар). При активации emergency_mode_handler

- 1) все цистерны немедленно отключаются;
- 2) блокируются все коммерческие операции;
- 3) фиксируется событие в истории;
- 4) выводится сообщение о вызове служб.

Код аварийного режима представлен на рисунке 6

```

1 def emergency_mode_handler(self):
2     print("ВНИМАНИЕ! Все цистерны будут заблокированы!")
3     confirm = input("Подтвердить аварию? (y/n): ").lower()
4
5     if confirm == 'y':
6         for cistern_id, cistern in self.cisterns.items():
7             if cistern['enabled']:
8                 cistern['enabled'] = False
9                 self.log_operation('EMERGENCY_DISABLE', f"
10                    Цистерна {cistern_id} заблокирована при аварии")
11
12             self.emergency_mode = True
13             self.log_operation('EMERGENCY', "Аварийная ситуация
14                 активирована")

```

Рисунок 6 – Аварийный режим

Выход из аварийного режима возможен только вручную, при этом цистерны не включаются автоматически — требование ТЗ выполнено строго.

2.6 Сохранение данных

Все данные сохраняются в файл station_data.json в формате JSON. Структура включает:

- 1) состояние цистерн;
- 2) баланс и статистику;
- 3) историю операций;
- 4) флаг аварийного режима.

Код сохранение данных изображен на рисунке 7

```

1 def save_data(self):
2     data = {
3         'cisterns': self.cisterns,
4         'balance': self.balance,
5         'stats': self.stats,
6         'history': self.history,
7         'emergency_mode': self.emergency_mode
8     }
9     with open('station_data.json', 'w', encoding='utf-8') as
f:
10        json.dump(data, f, ensure_ascii=False, indent=2)

```

Рисунок 7 – Сохранение данных

Методы save_data и load_data обеспечивают полное восстановление состояния программы после перезапуска.

3 Тестирование

В процессе разработки и после её завершения была проведена всесторонняя проверка функциональности системы. Тестирование включало следующие основные сценарии использования:

Было проведено комплексное тестирование всех функций системы. Проверялась корректность основных операций: обслуживания клиентов (полный цикл от выбора колонки до фиксации продажи), пополнения цистерн с валидацией объёмов и перекачки топлива между цистернами одного типа.

Система устойчива к некорректному вводу — обрабатывает нечисловые значения, отрицательные величины и выбор несуществующих пунктов. Аварийный режим корректно блокирует все операции, а после его отключения цистерны остаются заблокированными, как требует ТЗ.

Механизмы сохранения данных работают надёжно: состояние полностью восстанавливается после перезапуска. Автоматическое отключение цистерн при низком уровне топлива выполняется корректно. Все тесты подтвердили соответствие системы техническому заданию.

4 Заключение

Разработан прототип системы управления АЗС на Python, полностью соответствующий ТЗ. Реализованы:

- 1) полный цикл обслуживания клиентов;
- 2) управление запасами с автоматическим контролем уровня;
- 3) операции пополнения и перекачки топлива;
- 4) аварийный режим с ручным восстановлением;
- 5) сохранение состояния между запусками.

Архитектура на основе класса FuelStation обеспечивает связность и простоту поддержки. Код устойчив к ошибкам ввода, логирует все операции и легко расширяем.

Система готова для учебного использования и может служить основой для дальнейшего развития (веб-интерфейс, интеграция с учётными системами). Работа подтвердила возможность создания надёжных консольных систем управления на Python.