Минобрнауки России

Юго-Западный государственный университет

Факультет фундаментальной и прикладной информатики

Кафедра Программной инженерии

Лабораторная работа №2

По дисциплине: «Методология программной инженерии»

«Паттерн проектирования «Компоновщик»

Выполнил: студент группы ПО-51м

Журбенко В.А.

Проверила: к.т.н., профессор

Белова Т. М.

Курск – 2025 г.

Вариант 12. На городской автобазе имеется автотранспорт для перевозки пассажиров. Автотранспорт использует разные источники энергии: электричество, бензин, газ. Есть гибридные автобусы. Реализовать программный продукт, позволяющий автобазе сформировать список автомобилей с разными источниками энергии.

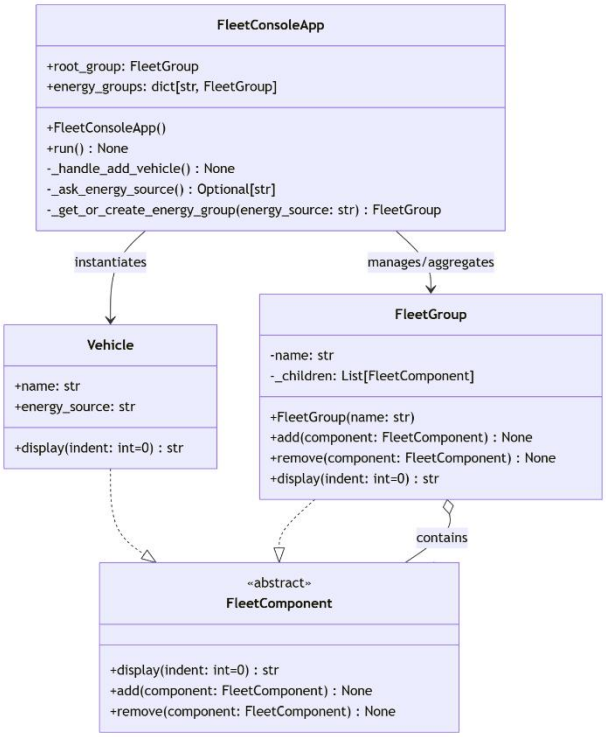

Рисунок 1 – UML-диаграмма

Код программы:

```
from __future__ import annotations


from abc import ABC, abstractmethod

from dataclasses import dataclass

from typing import List, Optional


class FleetComponent(ABC):
    """Base component for fleet items."""


    @abstractmethod
    def display(self, indent: int = 0) -> str:
        """Return a string representation of the component."""
```

```
    def add(self, component: "FleetComponent") -> None:
        raise NotImplementedError("Cannot add child to a leaf component")

    def remove(self, component: "FleetComponent") -> None:
        raise NotImplementedError("Cannot remove child from a leaf component")


class FleetGroup(FleetComponent):
    """Composite that can contain other components."""

    def __init__(self, name: str) -> None:
        self.name = name
        self._children: List[FleetComponent] = []

    def add(self, component: FleetComponent) -> None:  # type: ignore[override]
        self._children.append(component)

    def remove(self, component: FleetComponent) -> None:  # type: ignore[override]
        self._children.remove(component)

    def display(self, indent: int = 0) -> str:
        lines = [" " * indent + f"{self.name}:"]
        for child in self._children:
            lines.append(child.display(indent + 2))
        return "\n".join(lines)


@dataclass
class Vehicle(FleetComponent):
    """Leaf node representing a vehicle."""

    name: str
    energy_source: str

    def display(self, indent: int = 0) -> str:
        return " " * indent + f"{self.name} ({self.energy_source})"
```

```
class FleetConsoleApp:
    """Simple console interface to manage the bus fleet."""

    def __init__(self) -> None:
        self.root_group = FleetGroup("Автопарк")
        self.energy_groups: dict[str, FleetGroup] = {}

    def run(self) -> None:
        print("Добро пожаловать в систему управления автопарком!")
        while True:
            print("\nВыберите действие:")
            print("1. Добавить автобус")
            print("2. Показать структуру автопарка")
            print("3. Выйти")
            choice = input("> ").strip()

            if choice == "1":
                self._handle_add_vehicle()
            elif choice == "2":
                print("\n" + self.root_group.display())
            elif choice == "3":
                print("До свидания!")
                break
            else:
                print("Неизвестная команда. Попробуйте снова.")

    def _handle_add_vehicle(self) -> None:
        name = input("Введите название автобуса: ").strip()
        if not name:
            print("Название автобуса не может быть пустым.")
            return

        energy_source = self._ask_energy_source()
        if energy_source is None:
            return
```

```
        group = self._get_or_create_energy_group(energy_source)
        group.add(Vehicle(name=name, energy_source=energy_source))
        print(f"Автобус '{name}' с источником энергии '{energy_source}'
добавлен в автопарк.")

    def _ask_energy_source(self) -> Optional[str]:
        sources = [
            "электричество",
            "бензин",
            "газ",
            "гибрид",
        ]
        print("Выберите источник энергии:")
        for idx, source in enumerate(sources, start=1):
            print(f"{idx}. {source}")
        choice = input("> ").strip()
        if not choice.isdigit():
            print("Нужно ввести номер варианта.")
            return None
        index = int(choice) - 1
        if index < 0 or index >= len(sources):
            print("Нет такого источника энергии.")
            return None
        return sources[index]

    def _get_or_create_energy_group(self, energy_source: str) -> FleetGroup:
        if energy_source not in self.energy_groups:
            group = FleetGroup(energy_source.capitalize())
            self.energy_groups[energy_source] = group
            self.root_group.add(group)
        return self.energy_groups[energy_source]

def main() -> None:
    app = FleetConsoleApp()
    app.run()
```

```
if __name__ == "__main__":
    main()
```

Результат работы программы:

```
Выберите действие:
1. Добавить автобус
2. Показать структуру автопарка
3. Выйти
> 1
Введите название автобуса: ЗИЛ
Выберите источник энергии:
1. электричество
2. бензин
3. газ
4. гибрид
> 2
Автобус 'ЗИЛ' с источником энергии 'бензин' добавлен в автопарк.

Выберите действие:
1. Добавить автобус
2. Показать структуру автопарка
3. Выйти
> 1
Введите название автобуса: Altaya Atlas
Выберите источник энергии:
1. электричество
2. бензин
3. газ
4. гибрид
> 1
Автобус 'Altaya Atlas' с источником энергии 'электричество' добавлен в автопарк.
```

```
Выберите действие:
1. Добавить автобус
2. Показать структуру автопарка
3. Выйти
> 2

Автопарк:
  Электричество:
    Ikarus (электричество)
    Altaya Atlas (электричество)
  Гибрид:
    Mercedes-Benz (гибрид)
    Volkswagen (гибрид)
  Бензин:
    Skoda (бензин)
    ЗИЛ (бензин)
  Газ:
    ЗИС (газ)
```