Пугачев Владислав ИС-45

**Практическое занятие №8. Применение отладочных классов в проекте**

Цель работы:

Научиться использовать отладочные классы для улучшения качества кода и упрощения процесса разработки.

Выполненные задачи

1. Создание иерархии классов

Была создана следующая иерархия классов:

Automobile - абстрактный базовый класс

Truck - класс-наследник (Грузовик)

Car - класс-наследник (Легковой автомобиль)

Bus - класс-наследник (Автобус)

2. Реализация отладочного функционала

Создан класс Debuggable, который предоставляет:

Метод debug\_log() для логирования отладочной информации

Интеграцию с модулем logging Python

Автоматическое указание имени класса в сообщениях

3. Реализация методов и свойств

Базовый класс Automobile:

5 методов: start\_engine(), stop\_engine(), drive(), honk(), get\_info()

3 свойства: brand, model, is\_running

Классы-наследники (добавлено по 2 метода и 2 свойства):

Truck: методы load\_cargo(), unload(); свойства max\_load, current\_load

Car: методы add\_passenger(), remove\_passenger(); свойства doors, passengers

Bus: методы board\_passenger(), exit\_passenger(); свойства capacity, passengers

4. Переопределение методов

В каждом классе-наследнике переопределено по 3 метода:

start\_engine() - различное поведение запуска двигателя

drive() - различная манера движения

honk() - различные звуки сигнала

Метод get\_info() не переопределяется и остается общим для всех классов.

Код программы:

import logging

from abc import ABC, abstractmethod

# Настройка отладочного логирования

logging.basicConfig(level=logging.DEBUG, format='%(name)s - %(levelname)s - %(message)s')

logger = logging.getLogger('AutoDebug')

class Debuggable:

    """Базовый отладочный класс"""

    def debug\_log(self, message: str):

        logger.debug(f"[{self.\_\_class\_\_.\_\_name\_\_}] {message}")

class Automobile(ABC, Debuggable):

    def \_\_init\_\_(self, brand: str, model: str):

        super().\_\_init\_\_()

        self.\_brand = brand

        self.\_model = model

        self.\_is\_running = False

        self.debug\_log(f"Создан {brand} {model}")

    # Свойства

    @property

    def brand(self) -> str:

        return self.\_brand

    @property

    def model(self) -> str:

        return self.\_model

    @property

    def is\_running(self) -> bool:

        return self.\_is\_running

    # Методы

    def start\_engine(self):

        if not self.\_is\_running:

            self.\_is\_running = True

            self.debug\_log("Двигатель запущен")

        else:

            self.debug\_log("Двигатель уже работает")

    def stop\_engine(self):

        if self.\_is\_running:

            self.\_is\_running = False

            self.debug\_log("Двигатель остановлен")

        else:

            self.debug\_log("Двигатель уже остановлен")

    def drive(self):

        if self.\_is\_running:

            self.debug\_log("Едем")

        else:

            self.debug\_log("Сначала запустите двигатель")

    def honk(self):

        self.debug\_log("Сигналим!")

    def get\_info(self):

        return f"{self.\_brand} {self.\_model}"

    @abstractmethod

    def get\_type(self):

        pass

class Truck(Automobile):

    def \_\_init\_\_(self, brand: str, model: str, max\_load: int):

        super().\_\_init\_\_(brand, model)

        self.\_max\_load = max\_load

        self.\_current\_load = 0

    # Специфичные свойства

    @property

    def max\_load(self):

        return self.\_max\_load

    @property

    def current\_load(self):

        return self.\_current\_load

    # Переопределенные методы

    def start\_engine(self):

        if not self.\_is\_running:

            self.\_is\_running = True

            self.debug\_log("Дизельный двигатель грузовика запущен")

    def drive(self):

        if self.\_is\_running:

            self.debug\_log("Грузовик медленно едет")

        else:

            self.debug\_log("Запустите двигатель")

    def honk(self):

        self.debug\_log("ГРОМКИЙ сигнал грузовика!")

    # Специфичные методы

    def load\_cargo(self, weight: int):

        if self.\_current\_load + weight <= self.\_max\_load:

            self.\_current\_load += weight

            self.debug\_log(f"Загружено {weight} кг")

        else:

            self.debug\_log("Перегруз!")

    def unload(self):

        self.debug\_log(f"Разгружено {self.\_current\_load} кг")

        self.\_current\_load = 0

    def get\_type(self):

        return "Грузовик"

class Car(Automobile):

    def \_\_init\_\_(self, brand: str, model: str, doors: int):

        super().\_\_init\_\_(brand, model)

        self.\_doors = doors

        self.\_passengers = 0

    # Специфичные свойства

    @property

    def doors(self):

        return self.\_doors

    @property

    def passengers(self):

        return self.\_passengers

    # Переопределенные методы

    def start\_engine(self):

        if not self.\_is\_running:

            self.\_is\_running = True

            self.debug\_log("Двигатель автомобиля запущен")

    def drive(self):

        if self.\_is\_running:

            self.debug\_log("Автомобиль быстро едет")

        else:

            self.debug\_log("Запустите двигатель")

    def honk(self):

        self.debug\_log("Сигнал автомобиля")

    # Специфичные методы

    def add\_passenger(self):

        if self.\_passengers < 4:

            self.\_passengers += 1

            self.debug\_log("Пассажир добавлен")

        else:

            self.debug\_log("Нет мест")

    def remove\_passenger(self):

        if self.\_passengers > 0:

            self.\_passengers -= 1

            self.debug\_log("Пассажир вышел")

    def get\_type(self):

        return "Легковой автомобиль"

class Bus(Automobile):

    def \_\_init\_\_(self, brand: str, model: str, capacity: int):

        super().\_\_init\_\_(brand, model)

        self.\_capacity = capacity

        self.\_passengers = 0

    # Специфичные свойства

    @property

    def capacity(self):

        return self.\_capacity

    @property

    def passengers(self):

        return self.\_passengers

    # Переопределенные методы

    def start\_engine(self):

        if not self.\_is\_running:

            self.\_is\_running = True

            self.debug\_log("Двигатель автобуса запущен")

    def drive(self):

        if self.\_is\_running:

            self.debug\_log("Автобус едет по маршруту")

        else:

            self.debug\_log("Запустите двигатель")

    def honk(self):

        self.debug\_log("Сигнал автобуса")

    # Специфичные методы

    def board\_passenger(self):

        if self.\_passengers < self.\_capacity:

            self.\_passengers += 1

            self.debug\_log("Пассажир вошел")

        else:

            self.debug\_log("Автобус полон")

    def exit\_passenger(self):

        if self.\_passengers > 0:

            self.\_passengers -= 1

            self.debug\_log("Пассажир вышел")

    def get\_type(self):

        return "Автобус"

# Демонстрация работы

def demo():

    print("=== ДЕМОНСТРАЦИЯ РАБОТЫ КЛАССОВ ===\n")

    # Создание объектов

    truck = Truck("Volvo", "FH16", 20000)

    car = Car("Toyota", "Camry", 4)

    bus = Bus("Mercedes", "Sprinter", 20)

    vehicles = [truck, car, bus]

    # Демонстрация методов

    for vehicle in vehicles:

        print(f"\n--- Работа с {vehicle.get\_type()} ---")

        print(vehicle.get\_info())

        vehicle.start\_engine()

        vehicle.drive()

        vehicle.honk()

        vehicle.stop\_engine()

        # Специфичные методы

        if isinstance(vehicle, Truck):

            vehicle.load\_cargo(5000)

            vehicle.unload()

        elif isinstance(vehicle, Car):

            vehicle.add\_passenger()

            vehicle.remove\_passenger()

        elif isinstance(vehicle, Bus):

            vehicle.board\_passenger()

            vehicle.exit\_passenger()

if \_\_name\_\_ == "\_\_main\_\_":

    demo()

