МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №3

Специальность ПО11

Выполнил
Гулевич Е.А.
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
12.04.2025 г.

Брест 2025

Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Python

Задание 1. Преподаватель. Класс должен обеспечивать одновременное взаимодействие с несколькими объектами класса Студент. Основные функции преподавателя – ПроверитьЛабораторнуюРаботу, ПровестиКонсультацию, ПринятьЭкзамен, ВыставитьОтметку, ПровестиЛекцию.

Использованный паттерн: Наблюдатель

Выполнение:

Код программы:

```
class Student:
  def __init__(self, name):
    self.name = name

  def receive_notification(self, message):
    print(f"[Студент {self.name}] Уведомление: {message}")

  def submit_lab(self, lab_work):
    print(f"[Студент {self.name}] Сдал работу: '{lab_work}'")

  def take_exam(self):
    print(f"[Студент {self.name}] Сдаёт экзамен...")


class Teacher:
  def __init__(self):
    self.students = []

  def add_student(self, student):
    if student not in self.students:
      self.students.append(student)
      print(f"Студент {student.name} добавлен.")
    else:
      print(f"Студент {student.name} уже есть в списке.")

  def remove_student(self, student):
    if student in self.students:
      self.students.remove(student)
      print(f"Студент {student.name} удалён.")
    else:
      print(f"Студента {student.name} нет в списке.")

  def notify_students(self, message):
    for student in self.students:
      student.receive_notification(message)

  def check_lab_work(self, student, lab_work):
    if student in self.students:
      print(f"\n[Преподаватель] Проверяет работу '{lab_work}' у {student.name}...")
```

```
            grade = input("Введите оценку (1-5): ")
            student.receive_notification(f"Работа '{lab_work}' оценена на {grade}")
        else:
            print(f"Ошибка: {student.name} не найден в группе.")

    def conduct_consultation(self, topic):
        print(f"\n[Преподаватель] Начинает консультацию по теме: '{topic}'")
        self.notify_students(f"Началась консультация: '{topic}'")

    def take_exam(self):
        print("\n[Преподаватель] Экзамен начался!")
        self.notify_students("Экзамен начался! Приступайте.")
        for student in self.students:
            student.take_exam()

    def give_lecture(self, topic):
        print(f"\n[Преподаватель] Читает лекцию: '{topic}'")
        self.notify_students(f"Лекция: '{topic}'")


def main():
    teacher = Teacher()

    # Создаём студентов
    students = []
    n = int(input("Введите количество студентов: "))
    for i in range(n):
        name = input(f"Имя студента {i+1}: ")
        students.append(Student(name))
        teacher.add_student(students[-1])

    # Меню действий преподавателя
    while True:
        print("\n--- Меню ---")
        print("1. Проверить лабораторную работу")
        print("2. Провести консультацию")
        print("3. Принять экзамен")
        print("4. Провести лекцию")
        print("5. Выход")

        choice = input("Выберите действие (1-5): ")

        if choice == "1":
            print("\nСписок студентов:")
            for idx, student in enumerate(students, 1):
                print(f"{idx}. {student.name}")
            stud_idx = int(input("Выберите студента (номер): ")) - 1
            lab = input("Введите название работы: ")
            teacher.check_lab_work(students[stud_idx], lab)

        elif choice == "2":
```

```
            topic = input("Введите тему консультации: ")
            teacher.conduct_consultation(topic)

        elif choice == "3":
            teacher.take_exam()

        elif choice == "4":
            topic = input("Введите тему лекции: ")
            teacher.give_lecture(topic)

        elif choice == "5":
            print("Выход из программы.")
            break

        else:
            print("Ошибка: выберите 1-5.")


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

Спецификация ввода:

Введите количество студентов: <кол-во студентов>
Имя студента 1: <1-ый элемент>
Имя студента 2: <2-ой элемент>

Пример:

Введите количество студентов: 2
Имя студента 1: Егор
Студент Егор добавлен.
Имя студента 2: Антон
Студент Антон добавлен.

Спецификация вывода:

Меню
1. Проверить лабораторную работу
2. Провести консультацию
3. Принять экзамен
4. Провести лекцию
5. Выход
Выберите действие (1-5): <1-5>

Список студентов:
1. <1-ый студент>
2. <2-ой студент>
Выберите студента (номер): <номер студента>
Введите название работы: <название работы>

[Преподаватель] Проверяет работу <название работы> у <выбранного студента>
Введите оценку (1-5): <1-5>

[Студент <выбранный студент>] Уведомление: Работа <название работы> оценена на <1-5>

Пример:

--- Меню ---
1. Проверить лабораторную работу
2. Провести консультацию
3. Принять экзамен
4. Провести лекцию
5. Выход
Выберите действие (1-5): 1

Список студентов:
1. Егор
2. Антон
Выберите студента (номер): 1
Введите название работы: ОАИП 1

[Преподаватель] Проверяет работу 'ОАИП 1' у Егор...
Введите оценку (1-5): 4
[Студент Егор] Уведомление: Работа 'ОАИП 1' оценена на 4

Рисунки с результатами работы программы:

```
--- Меню ---
1. Проверить лабораторную работу
2. Провести консультацию
3. Принять экзамен
4. Провести лекцию
5. Выход
Выберите действие (1-5): 1

Список студентов:
1. Егор
2. Антон
Выберите студента (номер): 1
Введите название работы: ОАИП 1

[Преподаватель] Проверяет работу 'ОАИП 1' у Егор...
Введите оценку (1-5): 4
[Студент Егор] Уведомление: Работа 'ОАИП 1' оценена на 4
```

Задание 2. ДУ автомобиля. Реализовать иерархию автомобилей для конкретных производителей и иерархию средств дистанционного управления. Автомобили должны иметь присущие им атрибуты и функции. ДУ имеет три основные функции – удаленная активация сигнализации, удаленное открытие/закрытие дверей и удаленный запуск двигателя. Эти функции должны отличаться по своим характеристикам для различных устройств ДУ.

Использованный паттерн: Мост

Выполнение:

Код программы:

```
    from abc import ABC, abstractmethod


class RemoteControl(ABC):
    @abstractmethod
    def activate_alarm(self):
        pass

    @abstractmethod
    def lock_doors(self, lock: bool):
        pass

    @abstractmethod
    def start_engine(self):
        pass

    @abstractmethod
    def get_remote_type(self):
        pass


class BasicRemote(RemoteControl):
    def activate_alarm(self):
        print("Сигнализация активирована")

    def lock_doors(self, lock: bool):
        print(f"Двери {'закрыты' if lock else 'открыты'}")

    def start_engine(self):
        print("Двигатель запущен")

    def get_remote_type(self):
        return "Базовый"


class AdvancedRemote(RemoteControl):
    def activate_alarm(self):
        print("Сигнализация активирована с GPS отслеживанием")

    def lock_doors(self, lock: bool):
        print(f"Двери {'закрыты' if lock else 'открыты'} с автоматическим доводчиком")

    def start_engine(self):
        print("Двигатель запущен с автоматическим прогревом")

    def get_remote_type(self):
        return "Продвинутый"


class Car:
```

```
    def __init__(self, brand: str, model: str, remote: RemoteControl):
        self.brand = brand
        self.model = model
        self.remote = remote
        self.engine_on = False
        self.doors_locked = True

    def get_info(self):
        return f"{self.brand} {self.model} ({self.remote.get_remote_type()} ДУ)"

    def toggle_engine(self):
        if self.engine_on:
            print("Двигатель заглушен")
            self.engine_on = False
        else:
            self.remote.start_engine()
            self.engine_on = True

    def toggle_doors(self):
        self.doors_locked = not self.doors_locked
        self.remote.lock_doors(self.doors_locked)

    def activate_alarm(self):
        self.remote.activate_alarm()

    def get_status(self):
        print(
            f"Состояние: двигатель {'включен' if self.engine_on else 'выключен'}, "
            f"двери {'закрыты' if self.doors_locked else 'открыты'}"
        )


def create_remote():
    while True:
        print("\nВыберите тип пульта ДУ:")
        print("1. Базовый")
        print("2. Продвинутый")
        choice = input("Ваш выбор (1-2): ")

        if choice == "1":
            return BasicRemote()
        elif choice == "2":
            return AdvancedRemote()
        else:
            print("Неверный ввод. Попробуйте снова.")


def create_car():
    print("\nСоздание нового автомобиля")

    brand = input("Введите марку автомобиля: ")
```

```
    model = input("Введите модель автомобиля: ")

    remote = create_remote()

    return Car(brand, model, remote)


def car_control(car):
    while True:
        print(f"\nУправление автомобилем {car.get_info()}")
        print("1. Включить/выключить двигатель")
        print("2. Открыть/закрыть двери")
        print("3. Активировать сигнализацию")
        print("4. Показать состояние")
        print("5. Вернуться в меню")

        choice = input("Выберите действие (1-5): ")

        if choice == "1":
            car.toggle_engine()
        elif choice == "2":
            car.toggle_doors()
        elif choice == "3":
            car.activate_alarm()
        elif choice == "4":
            car.get_status()
        elif choice == "5":
            break
        else:
            print("Неверный ввод. Попробуйте снова.")


def main():
    cars = []

    while True:
        print("\nГлавное меню:")
        print("1. Добавить автомобиль")
        print("2. Управлять автомобилем")
        print("3. Выйти из программы")

        choice = input("Выберите действие (1-3): ")

        if choice == "1":
            car = create_car()
            cars.append(car)
            print(f"\nАвтомобиль {car.get_info()} успешно добавлен!")
        elif choice == "2":
            if not cars:
                print("Нет доступных автомобилей!")
                continue
```

```
            print("\nСписок автомобилей:")
            for i, car in enumerate(cars, 1):
                print(f"{i}. {car.get_info()}")

            while True:
                try:
                    car_num = int(input("Выберите номер автомобиля: ")) - 1
                    if 0 <= car_num < len(cars):
                        car_control(cars[car_num])
                        break
                    else:
                        print("Неверный номер. Попробуйте снова.")
                except ValueError:
                    print("Введите число!")
        elif choice == "3":
            print("Выход из программы.")
            break
        else:
            print("Неверный ввод. Попробуйте снова.")


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

Спецификация ввода:

Выберите действие (1-3): <1-3>

Создание нового автомобиля
Введите марку автомобиля: <марка авто>
Введите модель автомобиля: <модель авто>

Пример:

Выберите действие (1-3): 1

Создание нового автомобиля
Введите марку автомобиля: Ford
Введите модель автомобиля: Mustang

Спецификация вывода:

Главное меню:
1. Добавить автомобиль
2. Управлять автомобилем
3. Выйти из программы
Выберите действие (1-3): 1

Создание нового автомобиля
Введите марку автомобиля: Skoda
Введите модель автомобиля: Rapid

Выберите тип пульта ДУ:
1. Базовый
2. Продвинутый
Ваш выбор (1-2): 2

Автомобиль Skoda Rapid (Продвинутый ДУ) успешно добавлен!

Главное меню:
1. Добавить автомобиль
2. Управлять автомобилем
3. Выйти из программы
Выберите действие (1-3): 2

Список автомобилей:
1. Skoda Rapid (Продвинутый ДУ)
Выберите номер автомобиля: 1

Рисунки с результатами работы программы:

```
Главное меню:
1. Добавить автомобиль
2. Управлять автомобилем
3. Выйти из программы
Выберите действие (1-3): 1

Создание нового автомобиля
Введите марку автомобиля: Skoda
Введите модель автомобиля: Rapid

Выберите тип пульта ДУ:
1. Базовый
2. Продвинутый
Ваш выбор (1-2): 2

Автомобиль Skoda Rapid (Продвинутый ДУ) успешно добавлен!

Главное меню:
1. Добавить автомобиль
2. Управлять автомобилем
3. Выйти из программы
Выберите действие (1-3): 2

Список автомобилей:
1. Skoda Rapid (Продвинутый ДУ)
Выберите номер автомобиля: 1

Управление автомобилем Skoda Rapid (Продвинутый ДУ)
1. Включить/выключить двигатель
2. Открыть/закрыть двери
3. Активировать сигнализацию
4. Показать состояние
5. Вернуться в меню
Выберите действие (1-5): 1
Двигатель запущен с автоматическим прогревом
```

Задание 3. Проект «Пиццерия». Реализовать формирование заказ(а)ов, их отмену, а также повторный заказ с теми же самыми позициями.

Используемый паттерн : Команда

Код программы:

```
from dataclasses import dataclass, field
from enum import Enum
from typing import Dict, List


class PizzaType(Enum):
    MARGHERITA = "Маргарита"
    PEPPERONI = "Пепперони"
    BBQ = "Барбекю"
    SEAFOOD = "Морепродукты"
    VEGETARIAN = "Вегетарианская"
```

```
class PizzaSize(Enum):
    SMALL = "Маленькая"
    MEDIUM = "Средняя"
    LARGE = "Большая"


@dataclass
class Pizza:
    type: PizzaType
    size: PizzaSize
    price: float


@dataclass
class Order:
    order_id: int
    pizzas: List[Pizza] = field(default_factory=list)
    is_cancelled: bool = False

    def add_pizza(self, pizza: Pizza):
        self.pizzas.append(pizza)

    def cancel(self):
        self.is_cancelled = True

    def get_total(self) -> float:
        return sum(pizza.price for pizza in self.pizzas)

    def __str__(self):
        status = "Отменен" if self.is_cancelled else "Активен"
        pizzas = "\n".join(f"  - {pizza.size.value} {pizza.type.value}: ${pizza.price:.2f}" for pizza in self.pizzas)
        return f"Заказ #{self.order_id} ({status})\n" f"{pizzas}\n" f"Итого: ${self.get_total():.2f}"


class Pizzeria:
    def __init__(self):
        self._orders: Dict[int, Order] = {}
        self._next_order_id = 1
        self._price_list = {PizzaSize.SMALL: 8.99, PizzaSize.MEDIUM: 12.99, PizzaSize.LARGE: 16.99}

    def create_order(self) -> Order:
        order = Order(self._next_order_id)
        self._orders[self._next_order_id] = order
        self._next_order_id += 1
        return order

    def get_order(self, order_id: int) -> Order:
        return self._orders.get(order_id)

    def cancel_order(self, order_id: int) -> bool:
```

```
        order = self.get_order(order_id)
        if order and not order.is_cancelled:
            order.cancel()
            return True
        return False

    def reorder(self, order_id: int) -> Order:
        original = self.get_order(order_id)
        if not original:
            return None

        new_order = self.create_order()
        for pizza in original.pizzas:
            new_order.add_pizza(Pizza(pizza.type, pizza.size, pizza.price))
        return new_order

    def show_menu(self):
        print("\nМеню пиццерии:")
        print("Размеры:")
        for size in PizzaSize:
            print(f"  {size.value}: ${self._price_list[size]:.2f}")
        print("\nВиды пицц:")
        for pizza_type in PizzaType:
            print(f"  {pizza_type.value}")

    def create_pizza(self) -> Pizza:
        print("\nВыберите вид пиццы:")
        for i, pizza_type in enumerate(PizzaType, 1):
            print(f"{i}. {pizza_type.value}")

        type_choice = int(input("Ваш выбор (1-5): ")) - 1
        pizza_type = list(PizzaType)[type_choice]

        print("\nВыберите размер:")
        for i, size in enumerate(PizzaSize, 1):
            print(f"{i}. {size.value}")

        size_choice = int(input("Ваш выбор (1-3): ")) - 1
        pizza_size = list(PizzaSize)[size_choice]

        price = self._price_list[pizza_size]
        return Pizza(pizza_type, pizza_size, price)


def main():
    pizzeria = Pizzeria()

    while True:
        print("\nГлавное меню:")
        print("1. Создать новый заказ")
        print("2. Просмотреть активные заказы")
```

```
print("3. Отменить заказ")
print("4. Повторить заказ")
print("5. Выйти")

choice = input("Выберите действие (1-5): ")

if choice == "1":
    order = pizzeria.create_order()
    pizzeria.show_menu()

    while True:
        pizza = pizzeria.create_pizza()
        order.add_pizza(pizza)
        print(f"\nПицца добавлена в заказ #{order.order_id}")

        more = input("Добавить еще пиццу? (да/нет): ").lower()
        if more != "да":
            break

    print(f"\nЗаказ #{order.order_id} создан:")
    print(order)

elif choice == "2":
    print("\nАктивные заказы:")
    for order in pizzeria._orders.values():
        if not order.is_cancelled:
            print(order)
            print("-" * 30)

elif choice == "3":
    order_id = int(input("Введите номер заказа для отмены: "))
    if pizzeria.cancel_order(order_id):
        print(f"Заказ #{order_id} отменен")
    else:
        print("Не удалось отменить заказ (не найден или уже отменен)")

elif choice == "4":
    order_id = int(input("Введите номер заказа для повторения: "))
    new_order = pizzeria.reorder(order_id)
    if new_order:
        print(f"\nНовый заказ #{new_order.order_id} создан (копия #{order_id}):")
        print(new_order)
    else:
        print("Заказ не найден")

elif choice == "5":
    print("До свидания!")
    break

else:
    print("Неверный ввод. Пожалуйста, попробуйте снова.")
```

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

Спецификация ввода:

Выберите действие (1-5): <1-5>
Выберите вид пиццы:
Ваш выбор (1-5): <1-5>
Выберите размер:
Ваш выбор (1-5): <1-5>
Добавить еще пиццу? (да/нет): <да/нет>

Пример:

Выберите действие (1-5): 3
Выберите вид пиццы:
Ваш выбор (1-5): 2
Выберите размер:
Ваш выбор (1-5): 4
Добавить еще пиццу? (да/нет): нет

Спецификация вывода:

Главное меню:
1. Создать новый заказ
2. Просмотреть активные заказы
3. Отменить заказ
4. Повторить заказ
5. Выйти
Выберите действие (1-5): 1

Меню пиццерии:
Размеры:
  Маленькая: $8.99
  Средняя: $12.99
  Большая: $16.99

Виды пицц:
  Маргарита
  Пепперони
  Барбекю
  Морепродукты
  Вегетарианская

Выберите вид пиццы:
1. Маргарита
2. Пепперони
3. Барбекю

4. Морепродукты
5. Вегетарианская
Ваш выбор (1-5): 2

Выберите размер:
1. Маленькая
2. Средняя
3. Большая
Ваш выбор (1-3): 3

Пицца добавлена в заказ #1
Добавить еще пиццу? (да/нет): нет

Заказ #1 создан:
Заказ #1 (Активен)
  - Большая Пепперони: $16.99
Итого: $16.99

Пример:

```
Выберите действие (1-5): 1

Меню пиццерии:
Размеры:
  Маленькая: $8.99
  Средняя: $12.99
  Большая: $16.99

Виды пицц:
  Маргарита
  Пепперони
  Барбекю
  Морепродукты
  Вегетарианская

Выберите вид пиццы:
1. Маргарита
2. Пепперони
3. Барбекю
4. Морепродукты
5. Вегетарианская
Ваш выбор (1-5): 2

Выберите размер:
1. Маленькая
2. Средняя
3. Большая
Ваш выбор (1-3): 3

Пицца добавлена в заказ #1
Добавить еще пиццу? (да/нет): нет

Заказ #1 создан:
Заказ #1 (Активен)
  - Большая Пепперони: $16.99
Итого: $16.99
```

Вывод: приобрёл навыки применения паттернов проектирования при решении практических задач с использованием языка Python