📚 Краткая лекция: Введение в паттерны проектирования

🔷 Что такое паттерн проектирования?

Паттерн проектирования — это проверенное решение типовой проблемы в проектировании программного обеспечения.
Он **не является готовым кодом**, а описывает **общую структуру**, которую можно адаптировать под свои нужды.

🔷 Паттерн **Singleton** (Одиночка)

Цель: Гарантировать, что у класса **только один экземпляр** в программе, и предоставить к нему глобальную точку доступа.

Когда использовать:

- Для логгера
- Для базы данных (в простых случаях)
- Для настроек приложения

Как реализовать в Python:

```
class Logger:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance.log_file = "app.log"
        return cls._instance
```

> ✅ При любом количестве вызовов `Logger()` вы получите **один и тот же объект**.

🔷 Паттерн **Factory Method** (Фабричный метод)

Цель: Создавать объекты **без указания конкретного класса**, делегируя создание подклассам или фабрике.

Когда использовать:

- Когда тип объекта определяется во время выполнения
- Чтобы избежать прямого вызова конструкторов (`Dog()`, `Cat()` и т.д.)

Как реализовать:

```python
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Гав!"

class Cat(Animal):
    def speak(self):
        return "Мяу!"

class AnimalFactory:
    @staticmethod
    def create_animal(animal_type):
        if animal_type == "dog":
            return Dog()
        elif animal_type == "cat":
            return Cat()
        else:
            raise ValueError("Неизвестный тип животного")
```

> ✅ Теперь клиентский код не зависит от конкретных классов — только от фабрики.

🔷 Зачем это нужно?

- **Singleton** — контролирует количество экземпляров.
- **Factory Method** — упрощает расширение кода (добавили новый тип — обновили только фабрику).
- Оба паттерна делают код **гибче, чище и легче тестируемым**.

🧪 Практическое задание (60 минут): Система уведомлений с фабрикой и логгером

📌 Сценарий

Вы разрабатываете систему уведомлений для интернет-магазина.
Пользователи могут получать уведомления **разными способами**:

- По **email**
- Через **SMS**
- В **мобильное приложение**

Кроме того, все отправленные уведомления должны **записываться в лог**.
Но **логгер должен быть только один** в системе (чтобы не создавать множество файлов).

Ваша задача — реализовать:

1. **Фабрику уведомлений** (Factory Method)
2. **Единый логгер** (Singleton)

🎯 Часть 1: Классы уведомлений (15 минут)

Создайте базовый класс `Notification` и три дочерних:

```
class Notification:
    def send(self, message):
        pass

class EmailNotification(Notification):
    def send(self, message):
        return f"[Email] Отправлено: {message}"

class SMSNotification(Notification):
    def send(self, message):
        return f"[SMS] Отправлено: {message}"

class PushNotification(Notification):
    def send(self, message):
        return f"[Push] Отправлено: {message}"
```

🎯 Часть 2: Фабрика уведомлений (15 минут)

Создайте класс `NotificationFactory` с **статическим методом**:

```
class NotificationFactory:
    @staticmethod
    def create_notification(notification_type):
        # Реализуйте логику создания
        # Поддерживаемые типы: "email", "sms", "push"
        # При неизвестном типе — raise ValueError
```

🎯 Часть 3: Логгер как Singleton (15 минут)

Создайте класс `Logger`:

- Должен быть **только один экземпляр**.
- Иметь атрибут `log_entries` — список строк.
- Метод `log(message)` — добавляет сообщение в список.
- Метод `get_logs()` — возвращает копию списка логов.

> 💡 Используйте `__new__` для реализации Singleton.

🎯 Часть 4: Интеграция и демонстрация (15 минут)

Создайте функцию `send_notifications()`:

1. Получите **единственный экземпляр логгера**.
2. Создайте список типов уведомлений: `["email", "sms", "push", "email"]`.
3. Для каждого типа:
   - Создайте уведомление через фабрику.
   - Отправьте сообщение `"Ваш заказ подтверждён!"`.
   - Запишите результат в логгер.
4. Выведите все записи из лога.

📌 Пример вывода:

```
[Email] Отправлено: Ваш заказ подтверждён!
[SMS] Отправлено: Ваш заказ подтверждён!
[Push] Отправлено: Ваш заказ подтверждён!
[Email] Отправлено: Ваш заказ подтверждён!
```

> 💡 Убедитесь, что даже при нескольких вызовах `Logger()`, логгер **один и тот же** (проверьте через `id()` или добавьте запись до и после).

📝 Требования к сдаче

- Один файл `.py`.
- Классы: `Notification`, `EmailNotification`, `SMSNotification`, `PushNotification`, `NotificationFactory`, `Logger`.
- Функция `send_notifications()` с демонстрацией.
- Никакого `input()`.
- Singleton должен работать корректно (проверка: создайте два объекта `Logger` — они должны быть `is` друг другу).

💡 Подсказки

- В `Logger.__new__` проверяйте, создан ли `_instance`.
- В фабрике используйте `if/elif` или словарь с классами.
- В демонстрации:

```
logger1 = Logger()
logger2 = Logger()
print(logger1 is logger2)  # Должно быть True
```