



## Урок 6

# Поведенческие паттерны

Chain of Responsibility,  
Command, Mediator, Observer,  
Iterator, Interpreter, Memento,  
State, Strategy, Template Method,  
Visitor.

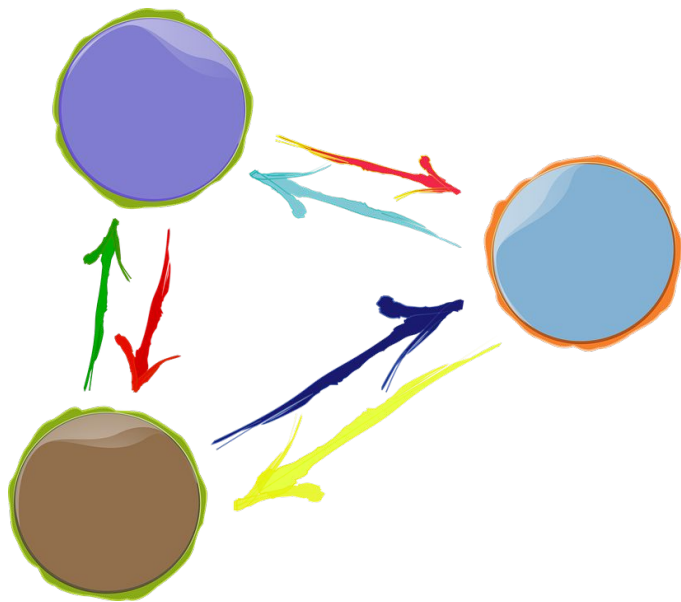
# План урока

- Поведенческие паттерны взаимодействия.
- Другие поведенческие паттерны.
- Практика.



# Поведенческие паттерны взаимодействия





# Поведенческие паттерны

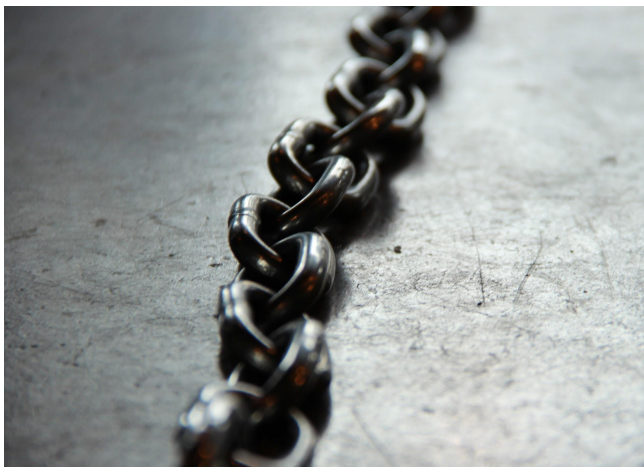
Поведенческие паттерны определяют алгоритмы и способы реализации взаимодействия различных объектов и классов.



# Поведенческие паттерны

- Цепочка ответственности.
- Команда.
- Посредник.
- Наблюдатель.

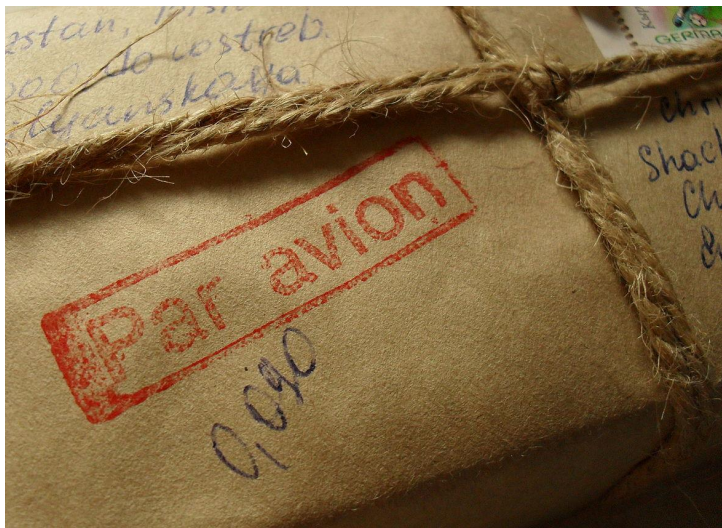




# Цепочка ОТВЕТСТВЕННОСТИ

Дать шанс обработать запрос  
нескольким участникам, связанным  
последовательно.





# Команда

Действие как объект. Позволит:

- Передавать как объект.
- Логировать действия.
- Ставить в очередь.
- Откатывать операции.



# Посредник

Определяет и координирует взаимодействие объектов.







# Наблюдатель

Создает механизм оповещения  
объектов об изменениях в других  
интересующих их объектах.



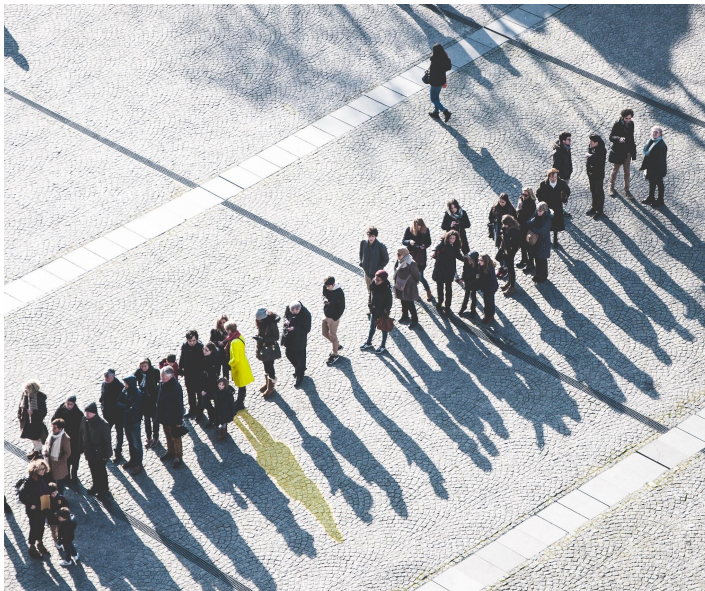
# Другие поведенческие паттерны



# Другие поведенческие паттерны

- Итератор.
- Интерпретатор.
- Хранитель.
- Состояние.
- Стратегия.
- Шаблонный метод.
- Посетитель.

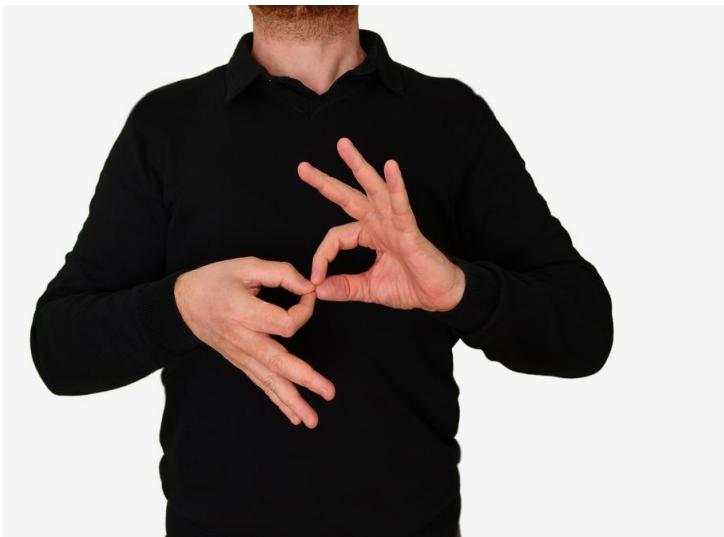




# Итератор

Предоставляет способ последовательного доступа ко всем элементам составного объекта, не раскрывая его внутреннего представления.





# Интерпретатор

Задаёт язык, определяет представление его грамматики, а также интерпретатор предложений этого языка.

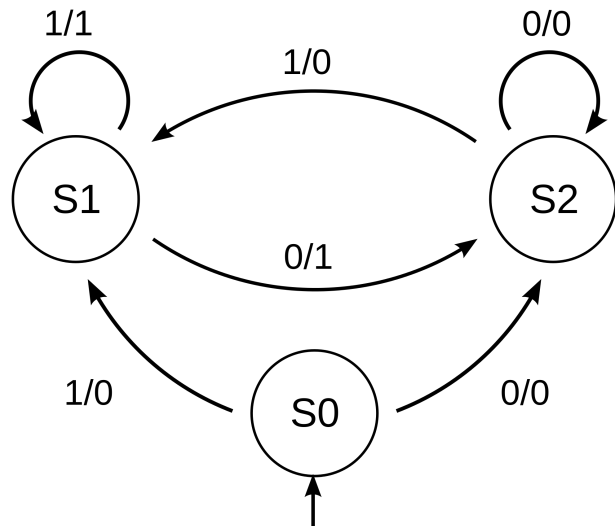




# Хранитель

Сохраняет и восстанавливает состояние объекта, не нарушая его инкапсуляции.





# Состояние

Позволяет объекту варьировать своё поведение в зависимости от внутреннего состояния.





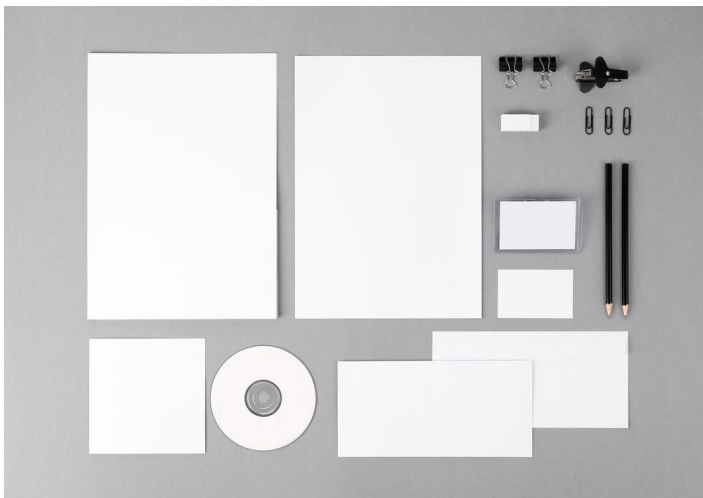


# Стратегия

Позволяет использовать  
взаимозаменяемые алгоритмы.







# Шаблонный метод

Определяет основу алгоритма и позволяет подклассам переопределить некоторые шаги алгоритма, не изменяя его структуру в целом.





# Посетитель

Позволяет определить новую операцию для иерархии классов, не изменяя сами классы.





# Практическое задание



**В этой самостоятельной работе тренируем умения:**

1. Выбирать подходящий поведенческий шаблон.
2. Применять поведенческие шаблоны в своём коде.

**Зачем:**

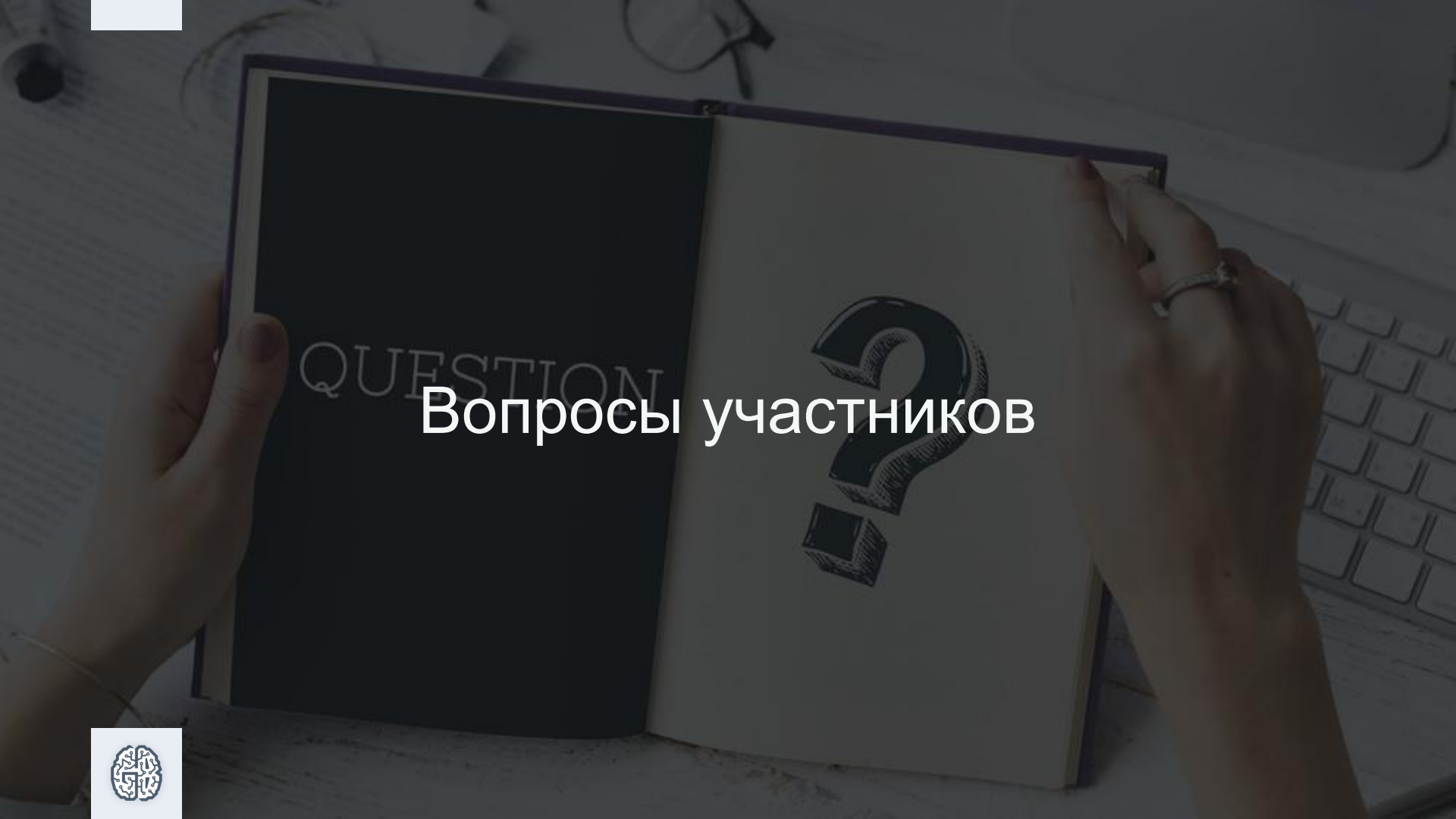
Для использования поведенческих шаблонов в своём коде.



## Последовательность действий:

1. Реализовать создание профиля студента (регистрация). Список студентов. Запись студента на курс.
2. Далее можно сделать всё или одно на выбор, применив при этом один из структурных паттернов, либо аргументировать, почему эти паттерны не были использованы:
  - Создать страницу для изменения курса. После изменения отправлять уведомления всем студентам на курсе по SMS, email (для имитации можно просто выводить сообщения в консоль). Также известно, что скоро способов уведомления будет больше.
  - Добавить возможность применять цикл `for` к объекту категории курса (в каждой итерации получаем курс) и объекта курса (в каждой итерации получаем студента). Например, `for student in course: ... for course in group`.
  - Создать API для курсов. По определённому адресу выводить не веб-страницу, а отдавать пользователю данные о списке курсов в формате `json`.
  - Улучшить логгер (или добавить, если его нет). Добавить в логгер возможность писать в файл, в консоль. Также известно, что в будущем вариантов сохранения может стать ещё больше.
  - Реализовать CBV (Class Based Views). Возможность создавать `view` в виде класса (по аналогии с Django). И убрать таким образом часть дублирования во `view`.





# Вопросы участников

