Видео-разбор подвига (решение смотреть только после своей попытки): <https://youtu.be/yXPFcDe6jco>

**Подвиг 8 (введение в паттерн миксинов - mixins).** Часто множественное наследование используют для наполнения дочернего класса определенным функционалом. То есть, с указанием каждого нового базового класса, дочерний класс приобретает все больше и больше возможностей. И, наоборот, убирая часть базовых классов, дочерний класс теряет соответствующую часть функционала.

Например, паттерн миксинов активно используют в популярном фреймворке Django.  В частности, когда нужно указать дочернему классу, какие запросы от клиента он должен обрабатывать (запросы типа GET, POST, PUT, DELETE и т.п.). В качестве примера реализуем эту идею в очень упрощенном виде, но сохраняя суть паттерна миксинов.

Предположим, что в программе уже существует следующий набор классов:

class RetriveMixin:

def get(self, request):

return "GET: " + request.get('url')

class CreateMixin:

def post(self, request):

return "POST: " + request.get('url')

class UpdateMixin:

def put(self, request):

return "PUT: " + request.get('url')

Здесь в каждом классе выполняется имитация обработки запросов. За GET-запрос отвечает метод get() класса RetriveMixin, за POST-запрос - метод post() класса CreateMixin, за PUT-запрос - метод put() класса UpdateMixin.

Далее, вам нужно объявить класс с именем GeneralView, в котором следует указать атрибут (на уровне класса):

allowed\_methods = ('GET', 'POST', 'PUT', 'DELETE')

для перечня разрешенных запросов. А также объявить метод render\_request со следующей сигнатурой:

def render\_request(self, request): ...

Здесь request - это словарь (объект запроса), в котором обязательно должны быть два ключа:

'url' - адрес для обработки запроса;  
'method' - метод запроса: 'GET', 'POST', 'PUT', 'DELETE' и т. д.

В методе render\_request() нужно сначала проверить, является ли указанный запрос в словаре request разрешенным (присутствует в списке allowed\_methods). И если это не так, то генерировать исключение командой:

raise TypeError(f"Метод {request.get('method')} не разрешен.")

Иначе, вызвать метод по его имени:

method\_request = request.get('method').lower()  # имя метода, малыми буквами

**Подсказка:** чтобы получить ссылку на метод с именем method\_request, воспользуйтесь магическим методом \_\_getattribute\_\_().

Для использования полученных классов, в программе объявляется следующий дочерний класс:

class DetailView(RetriveMixin, GeneralView):

    allowed\_methods = ('GET', 'PUT', )

Воспользоваться им можно, например, следующим образом (эти строчки в программе не писать):

view = DetailView()

html = view.render\_request({'url': 'https://stepik.org/course/116336/', 'method': 'GET'})

print(html)   # GET: https://stepik.org/course/116336/

Если в запросе указать другой метод:

html = view.render\_request({'url': 'https://stepik.org/course/116336/', 'method': 'PUT'})

то естественным образом возникнет исключение (реализовывать в программе не нужно, это уже встроено в сам язык Python):

AttributeError: 'DetailView' object has no attribute 'put'

так как дочерний класс DetailView не имеет метода put. Поправить это можно, если указать соответствующий базовый класс:

class DetailView(RetriveMixin, UpdateMixin, GeneralView):

    allowed\_methods = ('GET', 'PUT', )

Теперь, при выполнении команд:

view = DetailView()

html = view.render\_request({'url': 'https://stepik.org/course/116336/', 'method': 'PUT'})

print(html)

будет выведено:

PUT: https://stepik.org/course/116336/

Это и есть принцип работы паттерна миксинов.

P.S. В программе требуется объявить только класс GeneralView. На экран выводить ничего не нужно.

class RetriveMixin:

def get(self, request):

return "GET: " + request.get('url')

class CreateMixin:

def post(self, request):

return "POST: " + request.get('url')

class UpdateMixin:

def put(self, request):

return "PUT: " + request.get('url')

# здесь объявляйте класс GeneralView

class DetailView(RetriveMixin, UpdateMixin, GeneralView):

allowed\_methods = ('GET', 'POST', )