ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ  
«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

Специальность 09.02.07

«Информационные системы и программирование»

ТЕМА: Паттерны

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент гр. | ИС(ТП)-31 |  |  | | | |  |  | Соломатин А.Д. | | |
|  |  |  | *подпись* | | | |  |  | *Фамилия, И.О.* | | |
| Руководитель |  |  |  |  |  |  | | | |  | Брюханова И.Н. |
|  | *оценка* |  | *дата* |  |  | *подпись* | | | |  | *Фамилия, И.О.* |

Абакан, 2024

**Цель работы: https://github.com/Aleksey4ik6/Alekseyka**

**Задание №1**

Код для паттерна стратегия

2.Листинг программы:

from abc import ABC, abstractmethod

# Интерфейс стратегии

class Strategy(ABC):

@abstractmethod

def execute(self, data):

pass

# Конкретные стратегии

class StrategyA(Strategy):

def execute(self, data):

return sorted(data)

class StrategyB(Strategy):

def execute(self, data):

return list(reversed(data))

# Контекст

class Context:

def \_\_init\_\_(self, strategy: Strategy):

self.\_strategy = strategy

def set\_strategy(self, strategy: Strategy):

self.\_strategy = strategy

def execute\_strategy(self, data):

return self.\_strategy.execute(data)

# Использование

data = [3, 1, 4, 1, 5, 9]

context = Context(StrategyA())

print(context.execute\_strategy(data)) # [1, 1, 3, 4, 5, 9]

context.set\_strategy(StrategyB())

print(context.execute\_strategy(data)) # [9, 5, 1, 4, 1, 3]

3.Результаты



**Задание №2**

Код для паттерна одиночка

2.Листинг программы:

class SingletonMeta(type):

\_instances = {}

def \_\_call\_\_(cls, \*args, \*\*kwargs):

if cls not in cls.\_instances:

instance = super().\_\_call\_\_(\*args, \*\*kwargs)

cls.\_instances[cls] = instance

return cls.\_instances[cls]

class Singleton(metaclass=SingletonMeta):

def \_\_init\_\_(self, value):

self.value = value

# Использование

s1 = Singleton("Первый экземпляр")

s2 = Singleton("Второй экземпляр")

print(s1 is s2) # True

print(s1.value) # "Первый экземпляр"

print(s2.value) # "Первый экземпляр"

3.Результаты



**Задание №3**

Код для паттерна строитель

2.Листинг программы:

class Product:

def \_\_init\_\_(self):

self.parts = []

def add(self, part):

self.parts.append(part)

def show(self):

print(f"Product parts: {', '.join(self.parts)}")

# Интерфейс строителя

class Builder:

def \_\_init\_\_(self):

self.product = Product()

def reset(self):

self.product = Product()

def build\_part\_a(self):

self.product.add("Part A")

def build\_part\_b(self):

self.product.add("Part B")

def build\_part\_c(self):

self.product.add("Part C")

def get\_result(self):

return self.product

# Директор

class Director:

def \_\_init\_\_(self, builder: Builder):

self.builder = builder

def build\_minimal\_viable\_product(self):

self.builder.build\_part\_a()

def build\_full\_featured\_product(self):

self.builder.build\_part\_a()

self.builder.build\_part\_b()

self.builder.build\_part\_c()

# Использование

builder = Builder()

director = Director(builder)

# Создание минимального продукта

director.build\_minimal\_viable\_product()

product = builder.get\_result()

product.show()

# Создание полного продукта

builder.reset()

director.build\_full\_featured\_product()

product = builder.get\_result()

product.show()

3.Результаты

****