|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**ПРАКТИЧЕСКАЯ РАБОТА №3 по дисциплине**

**«Технологии и методы программирования»**

***(наименование дисциплины)***

**Тема практической работы   
«Шаблонный метод. Стратегия»**

Студент группы: БИСО-01-20 Маршев Л.А.

*(шифр учебной группы) (Фамилия И.О)*

Преподаватель: Лесько С.А.

*(Фамилия И.О)*

Москва, 2023

**Практическая работа №3**

**Стратегия**

Пример реализации паттерна Стратегия на языке Python:

class Strategy:

def Algorithm(self):

pass

class ConcreteStrategy1(Strategy):

def Algorithm(self):

print("ConcreteStrategy1")

class ConcreteStrategy2(Strategy):

def Algorithm(self):

print("ConcreteStrategy2")

class Context:

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

self.strategy = strategy

def Algorithm(self):

self.strategy.Algorithm()

context1 = Context(ConcreteStrategy1())

context1.Algorithm()

context2 = Context(ConcreteStrategy2())

context2.Algorithm()

В данном примере мы определяем интерфейс класса Strategy с методом Algorithm, который будут реализовывать все конкретные стратегии. Затем мы создаем два класса ConcreteStrategy1 и ConcreteStrategy2, которые наследуют интерфейс Strategy и реализуют свои алгоритмы в методе Algorithm.

Далее мы создаем класс Context, который принимает объект стратегии в качестве аргумента в своем конструкторе и использует его для вызова метода Algorithm. Создаем два экземпляра класса Context с разными объектами стратегии и вызываем метод Algorithm каждого из них.

**Шаблонный метод**

Пример реализации паттерна Шаблонный метод на языке Python:

class AbstractClass:

def TemplateMethod(self):

self.Operation1()

self.Operation2()

self.Operation3()

def Operation1(self):

pass

def Operation2(self):

pass

def Operation3(self):

pass

class ConcreteClass(AbstractClass):

def Operation1(self):

print("ConcreteClass - Operation1()")

def Operation2(self):

print("ConcreteClass - Operation2()")

class AnotherConcreteClass(AbstractClass):

def Operation1(self):

print("AnotherConcreteClass - Operation1()")

def Operation2(self):

print("AnotherConcreteClass - Operation2()")

def Operation3(self):

print("AnotherConcreteClass - Operation3()")

concrete = ConcreteClass()

concrete.TemplateMethod()

another\_concrete = AnotherConcreteClass()

another\_concrete.TemplateMethod()

В данном примере мы определяем абстрактный класс AbstractClass, который содержит метод TemplateMethod, в котором определен скелет алгоритма. Метод TemplateMethod вызывает методы Operation1, Operation2 и Operation3, которые могут быть переопределены в подклассах.

Затем мы создаем два конкретных класса ConcreteClass и AnotherConcreteClass, которые наследуют абстрактный класс AbstractClass и переопределяют методы Operation1, Operation2 и Operation3 в соответствии с конкретными требованиями этих классов.

Наконец, мы создаем экземпляры обоих классов и вызываем метод TemplateMethod для каждого из них. Благодаря использованию паттерна Шаблонный метод, мы можем легко изменять отдельные шаги алгоритма в подклассах, не затрагивая общей структуры.