|  |
| --- |
| 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 №4 по дисциплине**

**«Технологии и методы программирования»**

***(наименование дисциплины)***

**Тема практической работы   
«Итератор. Посетитель.»**

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 №4**

**Итератор**

Пример реализации паттерна Итератор на языке Python:

class Iterator:

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

self.collection = collection

self.index = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.index < len(self.collection):

item = self.collection[self.index]

self.index += 1

return item

raise StopIteration

# Пример использования

my\_list = [1, 2, 3, 4, 5]

my\_iterator = Iterator(my\_list)

for item in my\_iterator:

print(item)

В этом примере класс Iterator представляет итератор для коллекции, которую мы передаем в конструкторе. Он реализует методы \_\_iter\_\_ и \_\_next\_\_, которые позволяют итерироваться по элементам коллекции.

Метод \_\_iter\_\_ возвращает сам итератор, поэтому мы можем использовать его в цикле for.

Метод \_\_next\_\_ возвращает следующий элемент коллекции при каждой итерации. Если достигнут конец коллекции, он вызывает исключение StopIteration, чтобы сигнализировать о завершении итерации.

В приведенном примере мы создаем экземпляр итератора my\_iterator для списка my\_list и затем проходимся по каждому элементу списка с помощью цикла for, печатая каждый элемент.

**Посетитель**

Пример реализации паттерна Посетитель на языке Python:

class Visitor:

def visit\_elementA(self, element):

pass

def visit\_elementB(self, element):

pass

class Element:

def accept(self, visitor):

pass

class ElementA(Element):

def accept(self, visitor):

visitor.visit\_elementA(self)

class ElementB(Element):

def accept(self, visitor):

visitor.visit\_elementB(self)

class ConcreteVisitor(Visitor):

def visit\_elementA(self, element):

print("Visitor is visiting ElementA")

def visit\_elementB(self, element):

print("Visitor is visiting ElementB")

# Пример использования

element\_a = ElementA()

element\_b = ElementB()

visitor = ConcreteVisitor()

element\_a.accept(visitor)

element\_b.accept(visitor)

В этом примере паттерн Посетитель состоит из трех основных компонентов: Visitor, Element и конкретных реализаций ElementA, ElementB и ConcreteVisitor.

Visitor - это базовый класс для посетителя. Он определяет методы посещения (visit\_elementA, visit\_elementB), которые должны быть реализованы в конкретных посетителях.

Element - это базовый класс для элементов, которые могут быть посещены. Он определяет метод accept, который принимает посетителя в качестве аргумента и вызывает соответствующий метод visit\_elementX посетителя, передавая себя в качестве аргумента.

ElementA и ElementB - это конкретные реализации элементов. Они переопределяют метод accept, чтобы вызвать соответствующий метод посетителя для своего типа.

ConcreteVisitor - это конкретная реализация посетителя. Она наследуется от Visitor и реализует методы visit\_elementA и visit\_elementB для выполнения конкретных операций над элементами.

В приведенном примере мы создаем экземпляры элементов element\_a и element\_b, а также экземпляр конкретного посетителя visitor. Затем мы вызываем метод accept для каждого элемента, передавая посетителя. Каждый элемент вызывает соответствующий метод посетителя для своего типа. В данном случае, методы visit\_elementA и visit\_elementB просто выводят сообщения о посещении элементов.