**Практическая часть. №1 по билетам 1, 7, 13.**

Шаблон проектирования Singleton (Одиночка) гарантирует, что класс имеет только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру. Вот пример реализации на языке программирования Python:

class Singleton:

\_instance = None

def \_\_new\_\_(cls):

if not cls.\_instance:

cls.\_instance = super(Singleton, cls).\_\_new\_\_(cls)

return cls.\_instance

# Пример использования

instance1 = Singleton()

instance2 = Singleton()

print(instance1 is instance2) # Выведет True, так как это один и тот же объект

**Практическая часть. №1 по билетам 2, 5, 10.**

Вот пример реализации аналогичного функционала на языке Python:

class MyVector:

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

self.elements = []

# Метод для добавления элемента в конец вектора

def append(self, element):

self.elements.append(element)

# Метод для получения размера вектора

def size(self):

return len(self.elements)

# Метод для вывода элементов вектора

def display(self):

print("Elements of vector:", end=" ")

for element in self.elements:

print(element, end=" ")

print()

# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

# Создание объекта MyVector

my\_vector = MyVector()

# Добавление элементов в вектор

my\_vector.append(1)

my\_vector.append(2)

my\_vector.append(3)

# Вывод размера и элементов вектора

print("Size of vector:", my\_vector.size())

my\_vector.display()

Этот пример демонстрирует создание простого класса MyVector на Python, который имеет методы append, size и display. Метод append добавляет элемент в конец вектора, метод size возвращает размер вектора, а метод display выводит элементы вектора.

**Практическая часть. №1 по билетам 3, 8, 14.**

from abc import ABC, abstractmethod

# Абстрактный класс продукта

class Product(ABC):

@abstractmethod

def display(self):

pass

# Конкретные классы продуктов

class ConcreteProductA(Product):

def display(self):

return "Concrete Product A"

class ConcreteProductB(Product):

def display(self):

return "Concrete Product B"

# Абстрактный класс создателя

class Creator(ABC):

@abstractmethod

def factory\_method(self):

pass

def create\_product(self):

product = self.factory\_method()

return product

# Конкретные классы создателей

class ConcreteCreatorA(Creator):

def factory\_method(self):

return ConcreteProductA()

class ConcreteCreatorB(Creator):

def factory\_method(self):

return ConcreteProductB()

# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

creator\_a = ConcreteCreatorA()

product\_a = creator\_a.create\_product()

print(product\_a.display())

creator\_b = ConcreteCreatorB()

product\_b = creator\_b.create\_product()

print(product\_b.display())

В этом примере:

Product - это абстрактный класс продукта.

ConcreteProductA и ConcreteProductB - конкретные реализации продуктов.

Creator - абстрактный класс создателя с методом factory\_method, который подклассы реализуют для создания конкретных продуктов.

ConcreteCreatorA и ConcreteCreatorB - конкретные создатели, реализующие метод factory\_method для создания конкретных продуктов.

Пример использования создает экземпляры конкретных создателей (ConcreteCreatorA и ConcreteCreatorB) и использует их для создания соответствующих продуктов (ConcreteProductA и ConcreteProductB).