

ПРОГРАММА

ПРОФЕССИОНАЛЬНОЙ ПЕРЕПОДГОТОВКИ

**“Python-разработка для БПЛА: паттерны проектирования, API-интерфейсы и библиотеки для оптимизации решений”**

**Задание 6.**

**Реализация и использование поведенческих паттернов**

Преподаватель курса

**Максимов Егор Васильевич**

Выполнил

**Зубков Евгений Юрьевич**

**№ потока БПЛА\_256-1, группа 1**

**Москва 2024 г.**

**Задание 6. Реализация и использование поведенческих паттернов**

**Инструкция:**

Шаг 1. Реализовать паттерн Итератор. Создать класс MyIterator, реализовать методы **iter** и **next**.

Шаг 2. Реализовать паттерн Наблюдатель. Создать класс MyObserver с методом update. Создать класс MySubject, добавить метод register для подписки, метод notify для оповещения.

Шаг 3.Реализовать паттерн Шаблонный метод. Создать абстрактный класс MyBase с шаблонным методом template\_method. Создать классы MyClass1 и MyClass2, переопределить методы из базового класса.

Шаг 3. Протестировать Итератор, вызвав его в цикле for. Протестировать Наблюдатель - подписать наблюдателя, вызвать notify. Вызвать шаблонный метод в классах-наследниках.

**Решение:**

from abc import ABC, abstractmethod  
from typing import List  
  
  
# Реализация паттерна Итератор  
# В данном примере создается итератор для опроса дронов в рое дронов  
  
class Drone:  
 def \_\_init\_\_(self, drone\_list):  
 self.drone\_list = drone\_list  
 self.index = 0  
  
 def \_\_str\_\_(self):  
 return f'Current drone: {self.drone\_list}'  
  
  
class Iterator(ABC):  
 @abstractmethod  
 def has\_next(self) -> bool:  
 pass  
  
 @abstractmethod  
 def next(self) -> Drone:  
 pass  
  
  
class DroneIterator(Iterator):  
 def \_\_init\_\_(self, drone: List[Drone]):  
 self.\_drone = drone  
 self.\_index = 0  
  
 def next(self) -> Drone:  
 if self.has\_next():  
 drone = self.\_drone[self.\_index]  
 self.\_index += 1  
 return drone  
 else:  
 raise StopIteration  
  
 def has\_next(self) -> bool:  
 return self.\_index < len(self.\_drone)  
  
  
class DroneAggregator:  
 def \_\_init\_\_(self, drone\_count: int):  
 self.count = [Drone(i+1) for i in range(drone\_count)]  
  
 def amount\_of\_drones(self) -> int:  
 return len(self.count)  
  
 def create\_iterator(self) -> Iterator:  
 return DroneIterator(self.count)  
  
  
# Реализация паттерна Наблюдатель  
class MyObserver(ABC):  
 @abstractmethod  
 def update(self, message: str):  
 pass  
  
  
class MySubject:  
 def \_\_init\_\_(self):  
 self.\_observers = []  
  
 def register(self, observer: MyObserver):  
 self.\_observers.append(observer)  
  
 def notify(self, message: str):  
 for observer in self.\_observers:  
 observer.update(message)  
  
  
class AlertSystem(MyObserver):  
 def update(self, message: str):  
 print(f"Система предупреждения получила сообщение {message}")  
  
  
class AnalyseSystem(MyObserver):  
 def update(self, message: str):  
 print(f"Система анализа получила сообщение {message}")  
  
  
class DroneObservable(MySubject):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_altitude = 0  
  
 def set\_altitude(self, altitude: int):  
 self.\_altitude = altitude  
 self.notify(f"Высота дрона изменена на {altitude} метров")  
 if self.\_altitude > 500:  
 self.notify(f"Высота дрона превысила допустимую {self.\_altitude}м, ВНИМАНИЕ!")  
  
  
# Реализация паттерна Шаблонный метод  
class MyBase(ABC):  
 def template\_method(self):  
 self.step1()  
 self.step2()  
 self.step3()  
  
 @abstractmethod  
 def step1(self):  
 pass  
  
 @abstractmethod  
 def step2(self):  
 pass  
  
 @abstractmethod  
 def step3(self):  
 pass  
  
  
class MyClass1(MyBase):  
 def step1(self):  
 print("MyClass1: Реализация шага 1")  
  
 def step2(self):  
 print("MyClass1: Реализация шага 2")  
  
 def step3(self):  
 print("MyClass1: Реализация шага 3")  
  
  
class MyClass2(MyBase):  
 def step1(self):  
 print("MyClass2: Реализация шага 1")  
  
 def step2(self):  
 print("MyClass2: Реализация шага 2")  
  
 def step3(self):  
 print("MyClass2: Реализация шага 3")  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Тестирование итератора  
 print("Тестирование паттерна Итератор:")  
 print("Собираем рой дронов, где количество дронов - 5:")  
 drone\_aggregator = DroneAggregator(5)  
 print("Опрос дронов:")  
 drone\_iterator = drone\_aggregator.create\_iterator()  
 while drone\_iterator.has\_next():  
 item = drone\_iterator.next()  
 print(f" {item} is available.")  
  
print('-'\*31)  
  
# Тестирование паттерна Наблюдатель  
print("Тестирование паттерна Наблюдатель:")  
alert\_system = AlertSystem()  
analyse\_system = AnalyseSystem()  
drone\_observable = DroneObservable()  
drone\_observable.register(alert\_system)  
drone\_observable.register(analyse\_system)  
drone\_observable.set\_altitude(100)  
drone\_observable.set\_altitude(600)  
  
print('-'\*31)  
  
# Тестирование паттерна Шаблонный метод  
print("Тестирование шаблонного метода:")  
my\_class1 = MyClass1()  
my\_class1.template\_method()  
print()  
my\_class2 = MyClass2()  
my\_class2.template\_method()  
  
print('-'\*31)