

ПРОГРАММА

ПРОФЕССИОНАЛЬНОЙ ПЕРЕПОДГОТОВКИ

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

**Задание 8.**

**Proxy и Secure Proxy: Применение паттерна Proxy для обеспечения контроля доступа к объектам и данных.**

Преподаватель курса

**Максимов Егор Васильевич**

Выполнил

**Зубков Евгений Юрьевич**

**№ потока БПЛА\_256-1, группа 1**

**Москва 2024 г.**

**Задание 8. Proxy и Secure Proxy: Применение паттерна Proxy для обеспечения контроля доступа к объектам и данных.**

**Инструкция:**

Шаг 1. Создать интерфейс класса SomeObject с методами, предоставляющими доступ к данным.

Шаг 2. Реализовать класс Proxy, который реализует интерфейс SomeObject. В его методах добавить проверку прав доступа перед вызовом методов реального объекта.

Шаг 3. Реализовать класс SecureProxy. Также реализовать интерфейс SomeObject. В его методах добавить дополнительные проверки безопасности.

**Решение:**

*"""  
Домашнее задание 8  
Шаг 1. Создать интерфейс класса SomeObject с методами, предоставляющими доступ к данным.  
Шаг 2. Реализовать класс Proxy, который реализует интерфейс SomeObject. В его методах добавить проверку прав доступа перед вызовом методов реального объекта.  
Шаг 3. Реализовать класс SecureProxy. Также реализовать интерфейс SomeObject. В его методах добавить дополнительные проверки безопасности.  
"""*from abc import ABC, abstractmethod  
  
# Создаем абстрактный класс SomeObject  
class SomeObject(ABC):  
 @abstractmethod  
 def get\_data(self):  
 pass  
  
 @abstractmethod  
 def set\_data(self, data):  
 pass  
  
  
# Создаем реальный объект RealObject, который реализует интерфейс SomeObject  
class RealObject(SomeObject):  
 def \_\_init\_\_(self):  
 self.\_data = None  
  
 def get\_data(self):  
 return self.\_data  
  
 def set\_data(self, data):  
 self.\_data = data  
  
  
# Создаем прокси-класс Proxy  
class Proxy(SomeObject):  
 def \_\_init\_\_(self, real\_object, user):  
 self.\_real\_object = real\_object  
 self.\_user = user  
  
 def get\_data(self):  
 if self.\_user.has\_permission("get\_data"):  
 return self.\_real\_object.get\_data()  
 else:  
 raise PermissionError("User does not have permission to get data")  
  
 def set\_data(self, data):  
 if self.\_user.has\_permission("set\_data"):  
 self.\_real\_object.set\_data(data)  
 else:  
 raise PermissionError("User does not have permission to set data")  
  
# Создаем класс Пользователя User  
class User:  
 def \_\_init\_\_(self, name, permissions):  
 self.name = name  
 self.permissions = permissions  
  
 def has\_permission(self, permission):  
 return permission in self.permissions  
  
  
# Создаем прокси-класс SecureProxy, который реализует интерфейс SomeObject с дополнительной проверкой безопасности  
class SecureProxy(SomeObject):  
 def \_\_init\_\_(self, real\_object, user):  
 self.\_real\_object = real\_object  
 self.\_user = user  
  
 def get\_data(self):  
 if self.\_user.has\_permission("get\_data") and self.\_check\_security():  
 return self.\_real\_object.get\_data()  
 else:  
 raise PermissionError("Security check failed or user does not have permission to get data")  
  
 def set\_data(self, data):  
 if self.\_user.has\_permission("set\_data") and self.\_check\_security():  
 self.\_real\_object.set\_data(data)  
 else:  
 raise PermissionError("Security check failed or user does not have permission to set data")  
  
 def \_check\_security(self):  
 # Дополнительная проверка безопасности, например, проверка двухфакторной аутентификации  
 # Здесь добавим пример проверки: пользователь должен быть "admin"  
 return self.\_user.name == "admin"  
  
  
  
# Создаем интерфейс DroneInterface  
class DroneInterface(ABC):  
 @abstractmethod  
 def fly(self):  
 pass  
  
 @abstractmethod  
 def land(self):  
 pass  
  
  
# Создаем реальный объект Drone и прокси-класс SecureDroneProxy  
class Drone(DroneInterface):  
 def fly(self):  
 print("Drone is flying")  
  
 def land(self):  
 print("Drone is landing")  
  
class SecureDroneProxy(DroneInterface):  
 def \_\_init\_\_(self, drone, user):  
 self.\_drone = drone  
 self.\_user = user  
  
 def fly(self):  
 if self.\_check\_access() and self.\_check\_security():  
 self.\_drone.fly()  
 else:  
 raise PermissionError("Access denied or security check failed")  
  
 def land(self):  
 if self.\_check\_access() and self.\_check\_security():  
 self.\_drone.land()  
 else:  
 raise PermissionError("Access denied or security check failed")  
  
 def \_check\_access(self):  
 return self.\_user.has\_permission("fly")  
  
 def \_check\_security(self):  
 return self.\_user.name == "admin"  
  
  
# Запуск главной функции  
def main():  
 # Создаем пользователя и проверяем права доступа  
 user1 = User("admin", ["get\_data", "set\_data", "fly"])  
 user2 = User("guest", ["get\_data"])  
  
 # Создаем реальный объект и прокси  
 real\_object = RealObject()  
 proxy = Proxy(real\_object, user1)  
 secure\_proxy = SecureProxy(real\_object, user1)  
  
 # Работа с данными через прокси  
 proxy.set\_data("some data")  
 print(proxy.get\_data())  
  
 try:  
 secure\_proxy.set\_data("secure data")  
 print(secure\_proxy.get\_data())  
 except PermissionError as e:  
 print(e)  
  
 # Работа с беспилотником через прокси  
 drone = Drone()  
 secure\_drone\_proxy = SecureDroneProxy(drone, user1)  
 secure\_drone\_proxy.fly()  
 secure\_drone\_proxy.land()  
  
 try:  
 secure\_drone\_proxy = SecureDroneProxy(drone, user2)  
 secure\_drone\_proxy.fly()  
 except PermissionError as e:  
 print(e)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()