ДИСЦИПЛИНА **Методы верификации и валидации характеристик программного обеспечения**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **информационных технологий**

КАФЕДРА **математического обеспечения и стандартизации информационных технологий**

(полное наименование кафедры)

ВИД УЧЕБНОГО МАТЕРИАЛА **Материалы для практических/семинарских занятий**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Петренко Александр Анатольевич**

(фамилия, имя, отчество)

СЕМЕСТР **3, 2023-2024**

(указать семестр обучения, учебный год)

Контракты на программный код
(контрактное программирование)

На основе изучения материала лекций по дисциплине «Методы верификации и валидации характеристик программного обеспечения» требуется выполнить следующее.

1. Запрограммировать взаимодействующую объектно-ориентированную систему (несколько объектов) на языке Python.
2. Логика спроектированной системы должна быть реализована внутри объектов. При этом должен быть объект приложения, который создает все объекты, запускает их взаимодействие и печатает состояние программы.
3. Требуется выделить ограничения на свойства и результаты поведения программы, а также описать контракты на объекты.
4. Взаимодействующую программную систему необходимо реализовать в соответствии с принципом SCOOP. Для этого каждый объект должен работать в бесконечном цикле и взаимодействовать с другими объектами (рекомендуется использовать пример со спящим парикмахером).
5. Обработка ошибок должна быть выполнена с помощью описания контрактов и их проверки

Создание объектов и их взаимодействие.

Реализуем три класса:

- Client — представляет клиента, который приходит в салон.
- Barber — представляет парикмахера, который стрижет клиентов.
- Barbershop — представляет салон, управляющий очередью клиентов и парикмахером.

Логика приложения.

- Логика будет заключаться в создании объекта **Barbershop**, который управляет взаимодействием между **Client** и **Barber**. Состояние программы (например, количество ожидающих клиентов) будет выводиться через печать в консоль.

Ограничения и контракты.

Определим следующие ограничения:

- Очередь клиентов не может быть отрицательной.
- Парикмахер не может обслуживать клиента, если никто не ждет.
- Клиент не может быть обслужен, если парикмахер занят.

Контракты будем проверять с помощью утверждений (assert), чтобы убедиться, что ограничения выполняются.

Применение принципа SCOOP.

Каждый объект будет работать в отдельном потоке, используя бесконечный цикл для взаимодействия. Потоки обеспечат конкурентное выполнение.

Обработка ошибок.

Проверка контрактов и обработка ошибок будет включать исключения для ситуаций, когда контракты нарушаются.

Листинг кода

```
import threading
import time
import random

class Client:
    def __init__(self, id):
        self.id = id

    def __str__(self):
        return f"Client {self.id}"

class Barber:
    def __init__(self, shop):
        self.shop = shop
        self.is_cutting = False

    def cut_hair(self):
        while True:
            client = self.shop.get_next_client()
            if client:
                print(f"{client} is getting a haircut.")
                self.is_cutting = True
                time.sleep(random.uniform(1, 3))  # Симуляция времени стрижки
                print(f"{client} is done with the haircut.")
                self.is_cutting = False
            else:
                print("No clients, barber is sleeping.")
                time.sleep(1)  # Парикмахер ждет нового клиента

class Barbershop:
    def __init__(self, max_clients):
        self.clients = []
        self.max_clients = max_clients
        self.lock = threading.Lock()

    def add_client(self, client):
        with self.lock:
            if len(self.clients) < self.max_clients:
                self.clients.append(client)
                print(f"{client} entered the barbershop.")
            else:
```

```
            print(f"{client} left because the shop is full.")

    def get_next_client(self):
        with self.lock:
            if self.clients:
                return self.clients.pop(0)
            return None

def client_generator(shop):
    client_id = 1
    while True:
        time.sleep(random.uniform(0.5, 2))  # Симуляция времени прихода нового клиента
        client = Client(client_id)
        shop.add_client(client)
        client_id += 1

def main():
    shop = Barbershop(max_clients=3)
    barber = Barber(shop)

    # Запуск потоков для парикмахера и клиентов
    barber_thread = threading.Thread(target=barber.cut_hair)
    client_thread = threading.Thread(target=client_generator, args=(shop,))

    barber_thread.start()
    client_thread.start()

    # Приложение будет работать в течение 20 секунд, затем завершится
    time.sleep(20)

    print("Closing the barbershop.")

if __name__ == "__main__":
    main()
```

- **Client**, **Barber**, и **Barbershop** — классы, моделирующие основные объекты системы.
- **Barbershop** управляет очередью клиентов, используя блокировку для обеспечения потокобезопасности.
- **Barber** выполняет стрижку в бесконечном цикле, периодически проверяя наличие клиентов.
- **client_generator** создает новых клиентов в случайные моменты времени.
- Программа выполняется в течение 20 секунд, после чего завершает работу.

Контракты и обработка ошибок:

Контракты можно добавить с помощью утверждений, например:

Листинг кода контракта

```
assert len(self.clients) >= 0, "Очередь клиентов не может быть отрицательной."
```

SCOOP:

Каждый объект (**Barber** и клиенты) выполняется в своем потоке, реализуя принцип SCOOP для конкурентного взаимодействия.